ELEC677 Assignment 2 Report

Raymond Cano

Code: https://goo.gl/eSxK9v

Writeup

1) Visualizing a CNN with CIFAR10

a) CIFAR10 Dataset

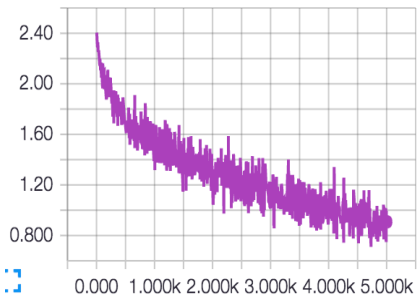
See code.

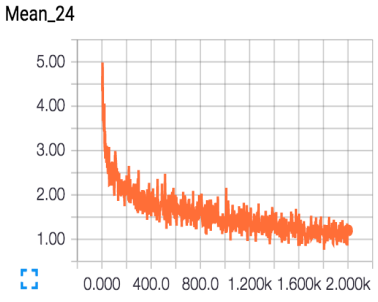
b) Train LeNet5 on CIFAR10

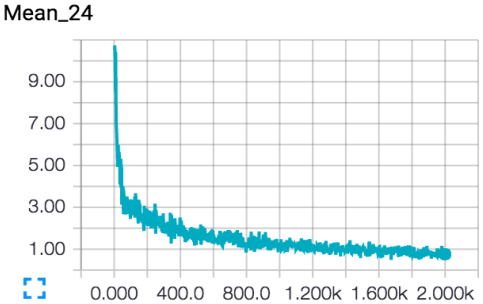
Training the LeNet5 on CIFAR10 didn't prove as fruitful as some of the other architectures I have tried before (Tensorflow Example, VGG Net, home made batches), however, it was quicker than other models due to the smaller size of the convolutional net. There was some improvement when Exponential Decay was applied to the learning rate. On occasion, a learning rate that started higher than 1E-4 would result in exploding gradients, causing a crash with NaN values.

The biggest improvement came from architecture improvements. The implementation of Batch Normalization and LeakyReLUs led to a 10 point accuracy improvement, from 47% to 57.3%. As one can see from the loss curves below, batch normalization moved the model towards convergence much faster than LeNet5, resulting in a similar training loss after 2000 iterations as LeNet5 had in 5000 training steps.

Figure 1: *The train loss curves for **a)** Standard LeNet5 with Xavier Initialzation and exponentially decaying learning rate and **b)** the model from a) with LeakyReLU units with alpha set to .1 and Batch Normalization layers.*







c) Visualize the Trained Network

Visualization was done using tensorboard's image summary after splitting the 32 channels of the first convolution's weights into 32 separate 5 pixel by 5 pixel images. In spite of the ability to visualize, it doesn't appear like the edge filters resemble those show in the example given the in lecture. A few theories I have for that are - The model trained poorly, which is caused by the inability to pick up edges - The model's initiali convolutional layer, which is 5x5, wasn't large enough to capture the edges across a 28x28 image. I've kind of done this all last minute, but will definitely investigate this theory later when I have free time. - The grayscale images don't provide as much contrast as color images, and thus, limit the ability to detect edges.

Images are attached below.

2) Visualizing and Understanding Convolutional Networks (A summary)

The Model

The paper introduces a method to visualize input features that excite feature maps the most, allowing us to understand how each layer discriminates on an image input. This is done through a deconvolutional net that reverses the actions of convolutional layers through indexed-tracked reverse max pooling approximation, rectifier units, and transposed filters. They attach these units to each convolutional layer to visualize the input coming in. The study then shows the images that excite these filters the most, giving us insite into the features that each map is excited by.

Takeawys from Visualization

The earlier layers of a convolutional net discriminate are excited by standard edges and edge patterns. However, later on, we see that the filter banks are excited most by particular features of a given class, or combination of similar classes. Obscuring these features through their Occlusion technique demonstrated the impact that features have on the accuracy of a convolutional net.

Debugging

The team used it's image exciting visualization to achieve a state-of-the-art score in ImageNet (at the time of experimentation) by understanding the shortcomings of the previous state-of-the-art. Though it did take a lot of work to visualize and understand the convolutions, the visualiations drive key insights into the workings of the architecture.

Transfer Learning

The team seems to stumble upon transfer learning, as they retrain a Softmax or SVM head attached to a pretrained imageNet classifier to achieve high scores in other image-related challenges. This feels like a textbook case of transfer learning, however, the term is never used.

3) Build and Train an RNN on MNIST

a) Training the RNN

The RNN was trained over many parameters. The most successful one reached a score of 95.21% training over 300,000 steps with an exponential decaying learning rate and 128 hidden units. It should be noted that the trianing loss still was yet to stabilize, as the RNN has much trouble converging. Tests of a larger number of hidden units was conducted (256) which scored lower by a nudge (95.18%), but had noticeably larger inconsistencies in test and train accuracy. The cost function used was the recommended softmax cross entropy with logits, and the optimizer was an Adam Optimizer. The learning rate was varied from 1e-3 to 1e-4, in addition to the use of the exponentially decaying learning rate. The ED learning rate stabilized the RNN a bit more, but it was still all over the place for the most part.

b) Using an LSTM or GRU

The LSTM and GRU had noticeably stabler training losses, and noticeably higher accuracy scores. The learning rate with these types of cells was of great importance, as using 1e-4 would lead the model to be trapped at a local optimum of 92% accuracy, while using the larger 1e-3 allowed enough jumping around to find optimums of around 98%. The LSTM and GRU both performed very well with 128 and 256 hidden units. The exponential decay learning rate proved the most succesful again,

giving the GRU based model the highest score of 98.06%, which also had 128 hidden units and trained for 100K steps. Models were trained either 100,000 or 200,000 steps. When using 256 hidden units, it was noted that the GRU performed worse over 200K steps due to overfitting. Images are attached to the bottom of this file.

c) Compare against the CNN

Speed

The RNN with a basic RNN cell trained incredibly quickly, handling 300,000 steps (~10,000 steps on the CNN) rather quickly. The LSTM and GRU cells trained much slower (around 3 minutes), though took no where as long as the 10-12 minutes that were had with the Convolutional Net.

Accuracy

The CNN proved much more accurate (by MNIST standards, we say 'much'), as the CNNs were able to score higher than the RNNs by a full percentage point.

Train Loss Stability

The CNN architectures converged in a much more stable manner than the RNN architectures while holding the number of epochs constant. This is largely in part due to the noted, notorious instibility of RNNs based on their tendency for gradient decay/explosion.

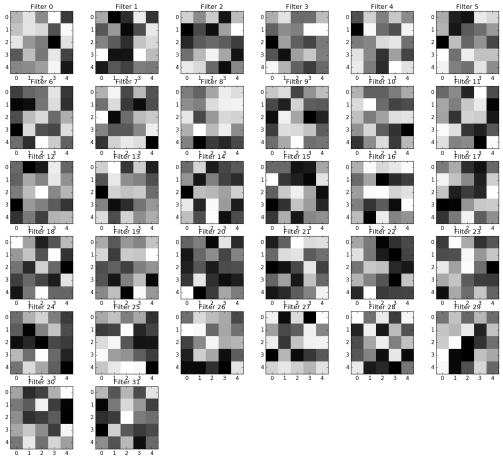
Images

CNN Visualizations

Test/Train Errors

Filters

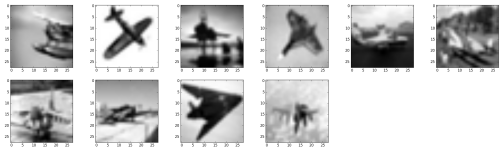
Below are visualizations of the filter banks for the 32 5 by 5filters generated in the first convolution of LeNet5. The are supposed to pick up basic edges in the images, and this is the most apparent in Figure 17, 29, and 14. However, the filters don't visualize as well as those shown in the assignment. I attribute this to the low dimensionality (5x5) of the filters, whereas most other first convolutions are kept around 7x7 or higher. The low accuracy of the model shows this as well, as a more accurate model would be able to develop filters that would discriminate on lines better.

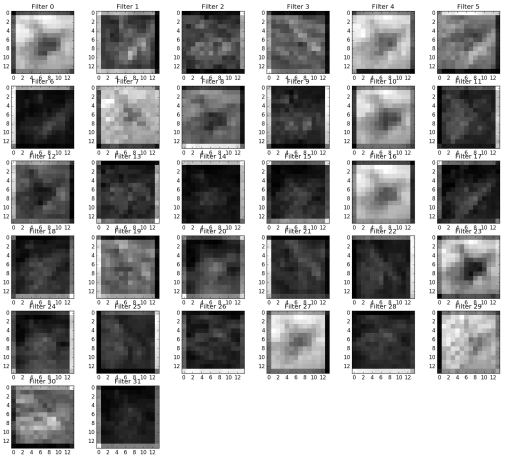


Activations

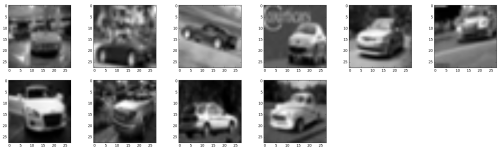
The plots below show the rendering of 10 test images from a given class on the left and (in lieu of activation *statistics*), I've presents activation maps of filter banks on the first level. The filter banks seen are the average across all 10 images across that class. Visually, you can see the patters of how that class activates the given filter map. Class 0 stands out in the vivid representation of the V like shape of the plane in the filter maps, while Class 6 activates the filters very strongly due to the diagonal pose of the frog.

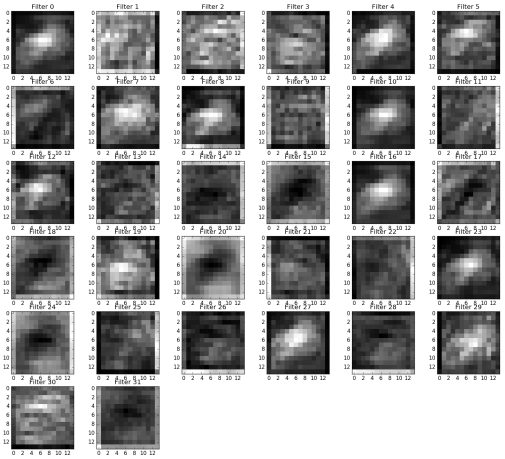
Class 0



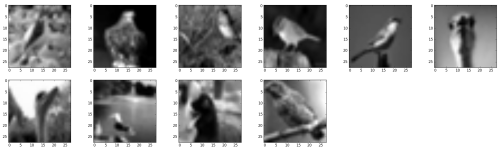


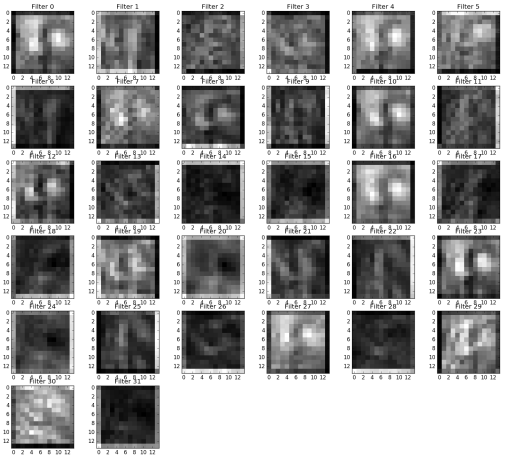
Class 1



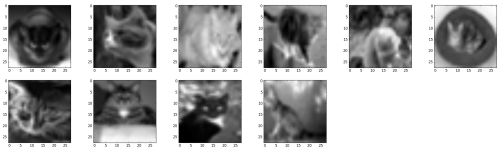


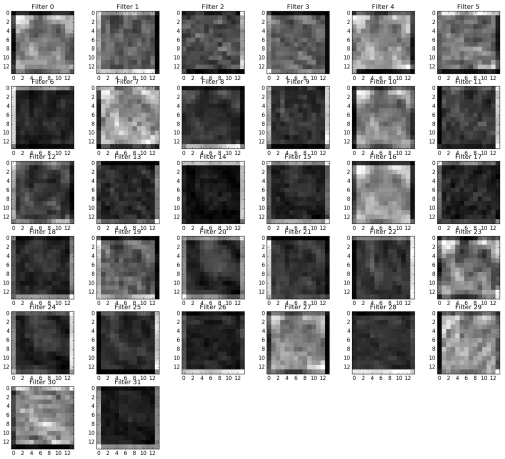
Class 2



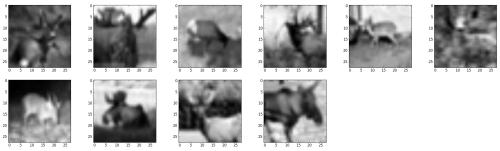


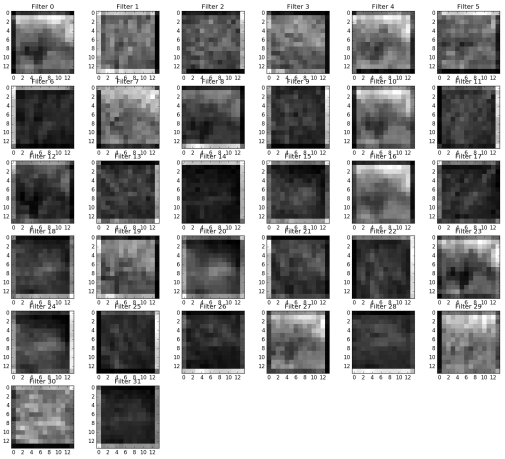
Class 3



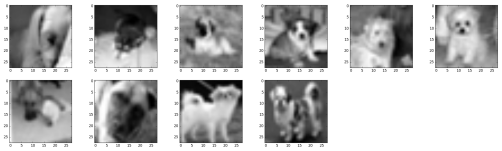


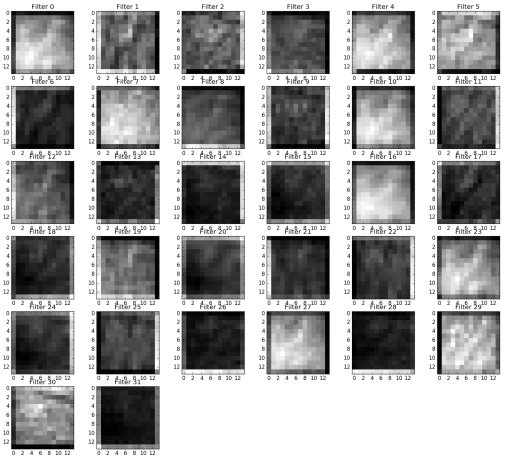
Class 4



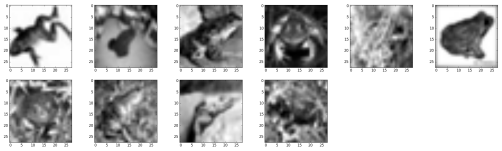


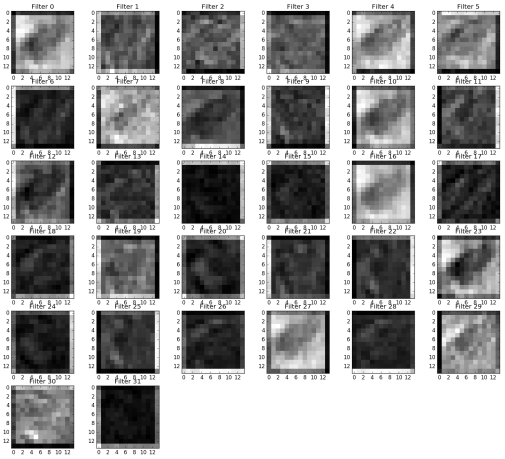
Class 5



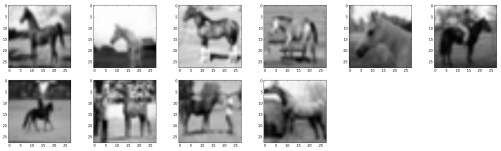


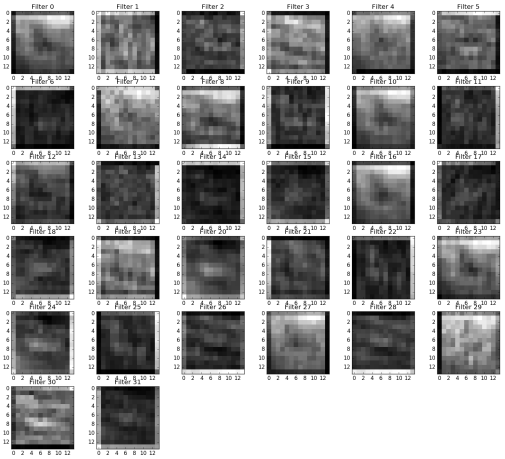
Class 6



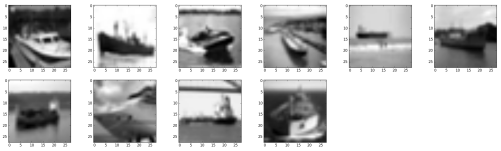


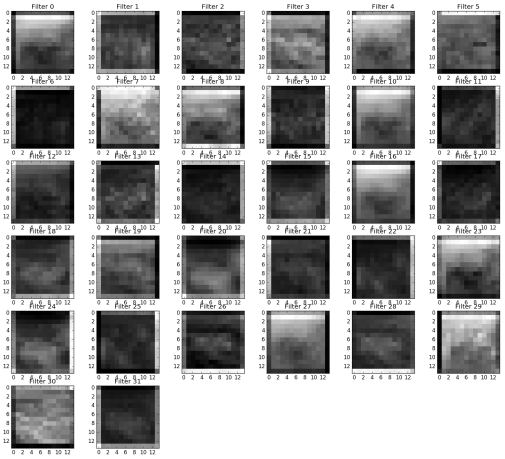
Class 7



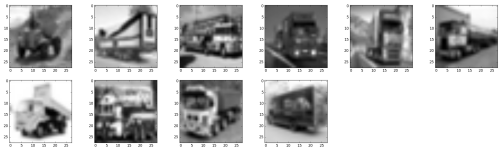


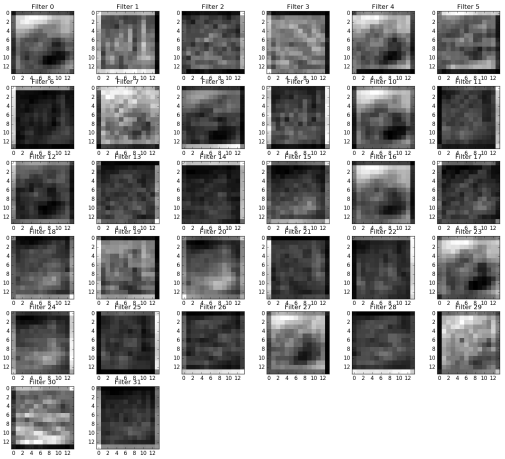
Class 8





Class 9



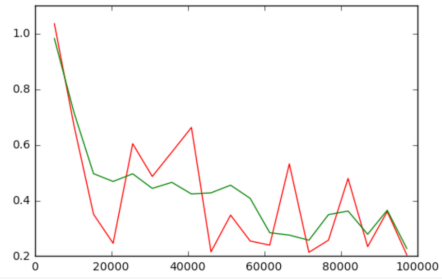


Activating Images

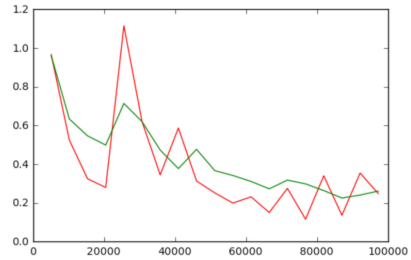
RNN Train/Test Errors

RNN Cell

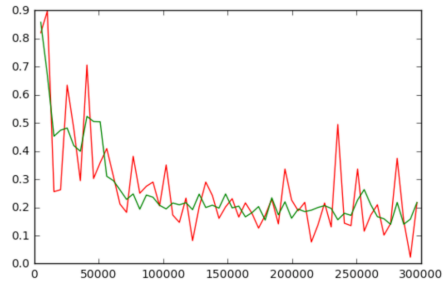
```
In [56]: plot_results('rnn_lreneg3_256_1e5')
```



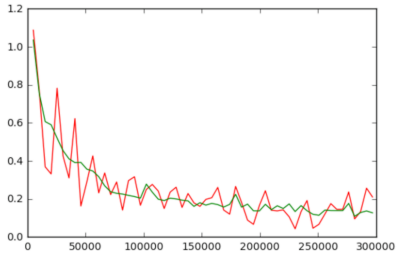
```
In [58]: plot_results('rnn_lrED_256_1e5')
```



```
In [60]: plot_results('rnn_lrED_256_3e5')
```

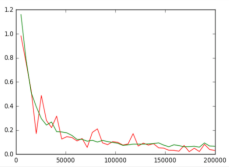


```
In [61]: plot_results('rnn_lrED_128_3e5')
```

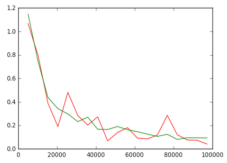


LSTM/GRU Cell

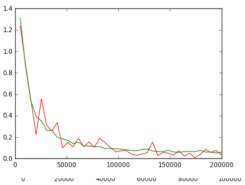
In [53]:
```
plot_results('gru_lrED_256_2e5')
```



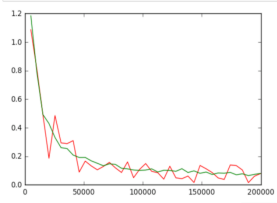
In [55]:
```
plot_results('gru_lreneg3_256_1e5')
```



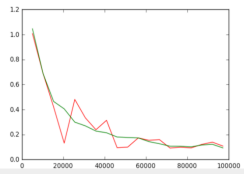
In [52]:
```
plot_results('gru_lrED_128_2e5')
```



In [51]:
```
plot_results('lstm_lrED_128_2e5')
```



In [50]:
```
plot_results('lstm_lrED_256_1e5')
```



In [49]:
```
plot_results('lstm_lreneg3_256_1e5')
```